DEFINING ALGORITHMS FOR PRODUCING, SOLVING AND CHECKING A SUDOKU INTERFACE WITH EFFICIENT COMPLEXITIES

Matriculation Number – 40542237

Module Number - SET08122

Author Email: 40542237@live.napier.ac.uk

Pages of interest – 3 (3, 4, 5)

Table of Contents

Abstract

The purpose of this research is to differentiate a variety of approaches to solve certain problems and compare them in fields of time and space complexity and to determine which way is the best to solve the Sudoku problem. This research will also deal with algorithms and data structures require to achieve the determined goal as stated above. This will include additional software and data architecture so that the Sudoku application will run with the highest efficiency with minimal time and space wastage. The paper will also deal with the language comparison and will state why the author had chosen Java-Maven as the development environment for the project.

1. Introduction

A Sudoku game is a board/grid based game usually consisting of a 9x9 architecture, thus having 81 cells. This paper will outline the three main requirements of the game and how to produce a solution to said needs. The first is to generate a random Sudoku grid that is almost always different. The second requirement is to allow the player to play the game by intentionally leaving out some cells. (The number of cells appearing on the screen will determine the difficulty of the puzzle). The third will deal with checking if the values are valid and if the puzzle has been solved. The additional requirements include undo-redo functionalities, game records and irregularly shaped or sized grids.

2. Implementations Evaluation and Strategies

The solution to the first requirement which is to generate a random Sudoku puzzle is influenced by the article on StackOverflow. The simple way of doing this is to take a random existing valid Sudoku board and shuffle it up and implement a backtracking algorithm. (User: Doc Brown, Sep 2, 2011). With a simple skim of this answer for creating unique Sudoku Grid on the website, one could state that a list (array) of random numbers can be a seed to produce the whole puzzle. The trick is to move three adjacent columns of the next row forward or backwards for the first three rows. Then switch up the order of the each three-columns in the sub-grid(square) so that a duplicate is not made on the 4th row and continue with the algorithm. Such method is used in commercial Sudoku Apps including ▻Sudoku (PeopleFun CG, LLC). The next step is to shuffle the columns and the rows, so that guessing the number pattern is basically near impossible. While doing this shuffle, it is easier to divide the number of rows in 3s and use the quotient and incrementing the for loop which will have the complexity of O (n/x) (where x = rows/3) and then access via currentRow + 2 or +3. Such procedure must be enwrapped in try/catch to prevent the application from crashing when producing irregular grids. Full Algorithm is explained in detail at zinhtun.github.algo.txt. (Appendix A). Such algorithm only needs 81 cell switching to produce a unique solution whereas, filling in the 1st, 5th and the 9th sub-grid (squares) and then the corners squares, which seemed to be the popular theory takes more than 300 search and randomization procedure. After the render, random cells must be taken out.

For the second step of the project, the interactive UI had been influenced by the C++ application written on GitHub (@flyingpeakock, Jun 12, 2021) (see Appendix B). The Change in color of the console and UTF – 8 is supported in IntelliJ, which is a convenience since the project is to be developed in Java.

The Third requirement consists of row and column checking using Tree datatype which will be discussed in details in section 3.3.

3. Features and Plan

3.1. Language of choice: Java

Java is chosen as the development language for the project and Maven is chosen as the standard packaging for the application. This allows one to use all the necessary packages in the engineering environment in the Object Oriented Way. Most projects during the research seem to manipulate memory using pointers based raw C or C extension (C++). For this project Java is chosen due to the versatility of OOP using Object Referencing which does not need any copy (not even the pointers, but reference counting is used) while passing through methods. Another advantage of using Java is the fact that Java has Automatic Garbage Collection that runs on a separate thread (M. Schoeberl, 2006). (C++ vs Java Memory Management: Appendix C).

3.2. Parallel Streaming

Another major advantage of Java and JVM (JAVA VIRTUAL MACHINE) is that this allows one to process the application as per power of hyper-threading of the CPU that is utilised. The only drawback of using parallel streaming is the data can be corrupted. All necessary processes are running parallel, thus changing the data of a variable during the loop can cause serious damage to one’s application. This method also produces process terminations in random orders. (Michael Müller, 2016). Parallel Streaming also is not thread safe, thus might skip some processes, ending up with null values in arrays if an insertion should happen within the loop. The saving grace though is the existence of CopyOnWriteArrayList <Template>, which is part of the Collections Framework and with such data-structures, safe parallel streaming is possible and thus is used in randomising the initial row of the Sudoku grid with faster write access.

3.3. Tree Architecture

A binary search tree is a very useful resource in building very complex systems without memory duplications and faster access time, promoting both time and space complexities. This is the architecture that is designed with author's prior knowledge from the module: Programming Fundamental. A tree divides the insertion into larger and smaller numbers thus have the time complexity of O log (n). A tree can also only have unique values, since a node cannot be inserted twice. It is either left (smaller) or right (larger). A three-nodes-tree can exist, but would not favor the situation of the Sudoku Grid Check. A grid check can be done by adding numbers that are already present in a row and a column as such if the number is already present in the not available list for the row, it doesn't need another insertion for the column saving a lot of space. The worst case scenario is that the tree became unbalanced, and even than it would still perform like a normal array for search, thus not a downside.

3.4. Other Requirements

As for the undo and redo, it would be wise to use persistence. Java has quite good support for buffer, stream and this file manipulation. A CSV would be really useful for the current case of Sudoku problem.

3.5. Datatypes not chosen

A normal ArrayList is not used as per project requirement the current tree can do everything and beyond its capabilities. The only advantage the ArrayList offers over the tree is it might use less memory at time compare to the tree since pointers and data exist. At other times an ArrayList will waste a lot of memory due to over allocation. A linked list is not used again as reference and scattered allocation is possible with the tree architecture.

4. Conclusion

To summarise, the use of Object-Oriented Programming Language like Java favours developer productivity by a huge margin when compared to pre-existing Sudoku projects/repositories in lower-level procedural programming languages. The increase in productivity results in focus of more interesting features like the use of efficient Trees and Nodes architectures and Parallel Streaming and algorithms for producing random Sudoku grid. All in all, with the research done and the project planned out with the components discussed, the Sudoku Application Supporting different levels of difficulties, sizes and shapes will be ready in a few months' time.

5. Appendices

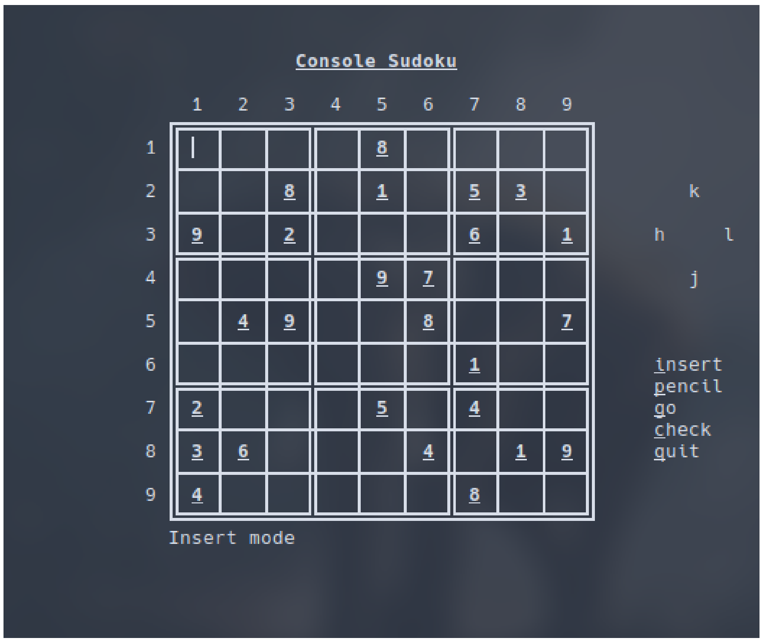
Appendix A

Text is copied from algo.txt at zinhtun.github.algo.txt.

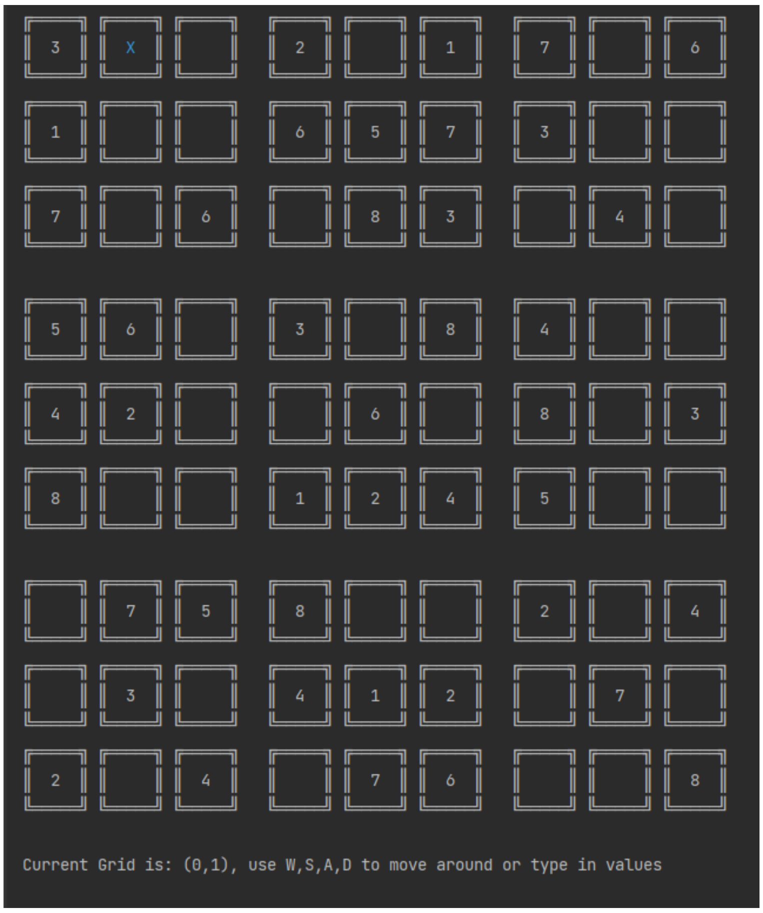
```
# First Generation

6 5 7  4 3 1  2 8 9 #let's say this is what we start off with

4 3 1  2 8 9  6 5 7

2 8 9  6 5 7  4 3 1


7 6 5  1 4 3  9 2 8

1 4 3  9 2 8  7 6 5

9 2 8  7 6 5  1 4 3


5 7 6  3 1 4  8 9 2

3 1 4  8 9 2  5 7 6

8 9 2  5 7 6  3 1 4


________________________________________________________
________________________________________________________



3 1 4  2 8 9  7 6 5

8 9 2  6 5 7  1 4 3

5 7 6  4 3 1  9 2 8


6 5 7  1 4 3  8 9 2

2 8 9  7 6 5  3 1 4

4 3 1  9 2 8  5 7 6


7 6 5  3 1 4  2 8 9

1 4 3  8 9 2  6 5 7

9 2 8  5 7 6  4 3 1

-----------------------------------------------------

// Algorithm step - 1
// Switch x, y

// Switch Columns
// 1st quadrant 0 with 2
// 3rd quadrant 6 with 8

// Switch Rows
/*

for +1 rows switch 0 - 2
for +2 rows switch 1 - 2
for +0 no switch.

x ++;
*/


__________________________________________________________
__________________________________________________________
```

Appendix B

@flyingpeakock's Design



```
                  Console Sudoku

        1   2   3   4   5   6   7   8   9
     1  |               8
     2          8       1       5   3               k
     3  9       2               6       1       h       l
     4                  9   7
     5      4   9           8           7           j
     6                          1
     7  2               5       4                insert
     8  3   6               4       1   9        pencil
     9  4                           8            go
                                                 check
     Insert mode                                 quit
```



```
[3] [X] [ ]   [2] [ ] [1]   [7] [ ] [6]
[1] [ ] [ ]   [6] [5] [7]   [3] [ ] [ ]
[7] [ ] [6]   [ ] [8] [3]   [ ] [4] [ ]

[5] [6] [ ]   [3] [ ] [8]   [4] [ ] [ ]
[4] [2] [ ]   [ ] [6] [ ]   [8] [ ] [3]
[8] [ ] [ ]   [1] [2] [4]   [5] [ ] [ ]

[ ] [7] [5]   [8] [ ] [ ]   [2] [ ] [4]
[ ] [3] [ ]   [4] [1] [2]   [ ] [7] [ ]
[2] [ ] [4]   [ ] [7] [6]   [ ] [ ] [8]

Current Grid is: (0,1), use W,S,A,D to move around or type in values
```

Figure: Author's Version of the Application

Appendix C

Java vs C/C++: Memory Manipulation/Management

Memory Management is a mandatory requirement for Raw C and C++, which are more low level and are operated closer to the machine without the use of a virtual machine or VM. These code are compiled directly to .obj, .asm and exe code (Windows) to be executed. As for Java, it uses JVM (Java Virtual Machine).

With C and C++ one have access to override security checks and run different memory checks possibly creating segmentation faults and leaks. This is a feature of said languages but is a double-edged sword as it also implies that one must always reference and dereference pointer and delete them in heap memory before the termination of the main process occurred. A simple developer mistake can cause memory overrides and leaks. C++ has a bit better support for this than C with smart pointers and automatic stream closers when termination happens, but it is still a good practice that developers have to destruct memory allocations (new keyword) after the application process has exited with return 0.

As of for Java, it enforces one to use classes and thus everything in Java is a child class or are derived from the Class Object. This enforcement of Object Orientation creates a very heavy space overhead but ensures one process and thread safety where references are forced to be used thus the application is only dependent upon reference counting. With the Automatic Garbage Collector Running on a separate thread, dead objects can easily be removed freeing space and making up for the heavy allocation for objects' memory at the start of the execution. Thus, one may never have seen the use of pointers in Java. Java also thus, enforce one to store object on the heap since reallocation is possible unlike the stack. This helps with developer productivity of a complex project since developers can now focus on sprint features in the application backlog rather than having to fight a low-level war with the computer.

6. References

Schoeberl, M. (2006, April). Real-time garbage collection for Java. In *Ninth IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC'06)* (pp. 9-pp). IEEE.

Müller, M. (2016). Java Lambdas and Parallel Streams. Apress

flyingpeakock, (2021). flyingpeakock/console_sudoku, <https://github.com/flyingpeakock/Console_sudoku>

Doc Brown, (2011). Stack Overflow, "How to generate Sudoku boards with unique solutions", < https://stackoverflow.com/questions/6924216/how-to-generate-sudoku-boards-with-unique-solutions>

PeopleFun CG, LLC (2023). ▻Sudoku (Version 7.21), https://apps.apple.com/us/app/sudoku/id366247306

Pfeffer, M., Ungerer, T., Fuhrmann, S., Kreuzinger, J., & Brinkschulte, U. (2004). Real-time garbage collection for a multithreaded Java microcontroller. *Real-Time Systems*, *26*, 89-106.